Application of Program Slicing

Home | Personal Data | Research Areas | Teaching

Application of Pr gram Slicing

Selection

Static Slicing

Dynamic Slicing

Slicing Application

Program Comprehension

A dynamic slice may be used to identify the parts of the program that contribute to the computation of the sel function for a given program execution (program input). Dynamic slicing may help to narrow down the prog part that contributes to the computation of the function of interest for a particular program input. A dynamic usually represented in the textual form, i.e., a dynamic slice is displayed to programmers in the form of highl statements in the original program or in the form of a subprogram by removing from the original program al statements that do not belong to the slice. Although dynamic slicing does reduce the size of a slice, it is still the programmer to analyze the dynamic slice. For some programs, there may be a relatively small decrease i size of a slice, or because of the significant size of the software system, the size of a slice may still be very la hard to comprehend. Programmers may still have difficulty understanding the program and its behavior. Si dynamic slices are only represented in the textual form, they may provide limited guidance in the process of understanding program execution. Therefore, it is important to devise methods that concentrate the progran attention only on the parts of the program and its execution that relate to the computation of the function of i It has been shown that dynamic program slicing is not only useful in software debugging but also in software maintenance, program comprehension, and software testing

Traditionally, in order to understand a program execution, a programmer uses conventional debuggers that s breakpoint facilities and stepwise program execution. Breakpoints allow a programmer to specify places in a program where the execution should be suspended. When a breakpoint is reached and the execution is susp the programmer can then examine various components of the program state and check the values of progran variables. Programmers may also execute a program in a step-wise manner in order to observe the program execution. Conventional debuggers, however, do not provide any to identify those program parts whose exe contribute to the computation of the function of interest. Using debuggers is an inefficient and time consumi approach for understanding program execution, especially when a programmer is interested in understandin those parts of the program execution that relate to a particular program function. The programmer may obse large amount of unrelated computation and it is frequently almost impossible for him/her to distinguish relat computations from non-related computations. In order to make the process of program understanding more efficient, it is important to reduce the amount of information a programmer sees and to focus his/her attentio related computations. Dynamic slicing techniques provide a means to prune away unrelated computation. I dynamic slice computation, different types of information are computed. For example, contributing actions a non-contributing actions are computed. This information is usually discarded after the slice computation. P Comprehension, as part of the software maintenance process is a crucial phase of system development. It is expected that a major share of systems development effort goes into modifying and extending preexisting sys about which we usually know little [4]. Through the increased complexity of software systems, their mainten becoming more and more aggravating. Change to a system may be necessitated for adaptive, perfective, cor or preventive reasons. Understanding the system, incorporating the change, and testing the system to ensure change had no united effect on the system are the three facets of software maintenance. Reverse engineering of program comprehension can be described as analyzing a subject system to (a) identify the system's compc and their interrelationships, (b) to create representations of a system in another form at a higher level of abst and (c) to understand the program execution and the sequence in which it occurred

Dynamic Slicing

[illegible] [illegible] [illegible] [illegible]

Dynamic Slicing

Selection

[illegible]

A dynamic program slice is this part of a program that "affects" the computation of a variable of interest duri
program execution on a specific program input. Dynamic program slicing refers to a collection of program s
methods that are based on program execution and may significantly reduce the size of a program slice becau
run-time information, collected during program execution, is used to compute program slices. Dynamic prog
slicing was originally proposed only for program debugging, but its application has been extended to progra
comprehension, software testing, and software maintenance. Different types of dynamic program slices, toge
with algorithms to compute them, have been proposed in the literature.

A static program slice consists of all statements in a program that may affect the value of variable *v* at some
As originally introduced, static slicing involves all possible program executions, i.e., the static slice preserve
computation of variable *v* at point *p* for all program inputs. However, in many situations one is interested in
that preserves the program's behavior for a specific program input, rather than that for all inputs. This type o
is referred to as *dynamic* slicing. The concept of dynamic program slicing was for the first time presented in
taking a particular program execution into account, dynamic slicing may significantly reduce the size of the s
compared to static slicing. During program execution it is possible to monitor different types of run-time
information that can be used during dynamic slice computation, e.g., addresses of array elements, addresses
elements of dynamic data structures that are referenced during program execution, etc. Most of the existing
slicing techniques have been proposed for sequential programs. But, the concept of dynamic slicing has bee
extended to distributed programs . Two major types of dynamic slicing have been proposed in the literature:
executable dynamic slicing and non-executable dynamic slicing. Informally, an executable dynamic slice is a
that can be executed and it preserves a value of a variable of interest. On the other hand, a non-executable sl
subprogram that contains statements that "influence" the variable of interest and, in most cases, it cannot be
executed.

A dynamic slice on the other hand overcomes the limitations of the static algorithms by being based on a par
program execution (program input). Therefore the dynamic slicing computation can utilize information abou
actual program flow for a particular program execution, which leads to an accurate handling of dynamic and
conditional language constructs and therefore to smaller program slices. One of the major drawbacks of dyn
slicing (compared with static slicing) is that it is necessary to identify relevant input conditions for which a d
slice should be computed. A commonly used approach to identify such input conditions is referred to as an
operational profile. It is a well-known concept that is frequently applied in testing and software quality assur
For the determination of the profile the relative frequencies of occurrence of the run types (inputs) are used,
expressed as fractions of the total of runs representing their probabilities.

Static Slicing

Static Slicing

Based on the original definition of Weiser [Wei84] a static program slice *S* consists of all statements in pro
that may affect the value of variable *v* at some point *p*. The slice is defined for a slicing criterion *C=(x, V)*, v
a statement in program *P* and *V* is a subset of variables in *P*. Given C, the slice consists of all statements in
potentially affect variables in *V* at position *x*. A static slicing includes all the statements that affect variable
set of *all* possible inputs at the point of interest. Static slices are computed by finding consecutive sets of ir
relevant statements, according to data and control dependencies.

The program dependence graph (PDG) was originally defined by Ottenstein and Ottenstein [Ott84] and lat
by Horwitz et al.[Hor88, Hor90, Rep88, and Rep89]. Data and control dependencies between nodes may f
program dependence graph, that can be used for the computation of static slices by traversing backwards al
edges of the program dependence graph from a point of interest. The program dependence graph is formec
combining all data and control dependencies that exist in the program. The static slice of a program with r
a variable *v* at a node *i*, consists of all nodes whose execution could possibly affect the value of the variable
node *i*. The static slice can be easily constructed with the PDG by traversing backwards along the edges of
program dependence graph starting at a node *i*. The nodes, which were visited during the traversal, constit
desired slice [Ott84].

The strength of static slicing lies in particular in the following areas: (a) the computation of a static slice is
cheap (compared to the dynamic slice) hence only the static analysis of the source code and no program exe
required, (b) it is useful if a user wants to gain a general understanding of the program parts that contribute
computation of a selected function with respect to all possible program executions

Best Available Copy